

```

CREATE TABLE School (
    Name          VARCHAR(256) NOT NULL,
    TotalEnrollment  INTEGER,
    State          VARCHAR(20),
    StateSalaryRank  INTEGER,
    EarlyCareerPay   REAL,
    MidCareerPay     REAL,
    STEMPercent     INTEGER,
    Type            VARCHAR(20),
    DegreeLength     VARCHAR(10),
    BoardCost        REAL,
    InStateTuition   REAL,
    OutStateTuition  REAL
    PRIMARY KEY(Name)
);

```

```

CREATE TABLE Diversity (
    Name          VARCHAR(256) NOT NULL,
    WomenCount    INTEGER,
    AIANCount     INTEGER,
    AsianCount    INTEGER,
    BlackCount    INTEGER,
    HispanicCount  INTEGER,
    PacificCount  INTEGER,
    WhiteCount    INTEGER,
    PRIMARY KEY(Name),
    FOREIGN KEY(Name) REFERENCES School(Name) ON DELETE CASCADE
);

```

```

CREATE TABLE State (
    StateName      VARCHAR(20) NOT NULL,
    Population     INTEGER,
    CrimeRate      REAL,
    HappinessScore  REAL,
    Code           CHAR(2),
    PRIMARY KEY(StateName)
);

```

```

CREATE TABLE Company (
    CompanyName    VARCHAR(256) NOT NULL,
    Industry        VARCHAR(100),
    Location        VARCHAR(256),
    EmployeeCount  INTEGER,
    PRIMARY KEY(CompanyName)
);

```

);

CREATE TABLE RateProf (

StudentID	INTEGER NOT NULL,
ProfessorName	VARCHAR(50),
SchoolName	VARCHAR(256),
State	CHAR(2),
StarRating	REAL,
DifficultyRating	REAL,
CourseID	VARCHAR(20),
DepartmentName	VARCHAR(256),
PRIMARY KEY(StudentID)	

);

CREATE TABLE Major (

Major	VARCHAR(256) NOT NULL,
Ranking	INTEGER,
TotalCount	INTEGER,
WomenCount	INTEGER,
Category	VARCHAR(256),
EmploymentCount	INTEGER,
MedianSalary	REAL,
PRIMARY KEY(Major)	

);

Code version of DDL Commands:

```
CREATE TABLE School (Name VARCHAR(256) NOT NULL, TotalEnrollment INTEGER,
State VARCHAR(20), StateSalaryRank INTEGER, EarlyCareerPay REAL,
MidCareerPay REAL, STEMPercent INTEGER, Type VARCHAR(20), DegreeLength
VARCHAR(10), BoardCost REAL, InStateTuition REAL, OutStateTuition REAL,
PRIMARY KEY(Name));
```

```
CREATE TABLE Diversity (Name VARCHAR(256) NOT NULL, WomenCount INTEGER,
AIANCount INTEGER, AsianCount INTEGER, BlackCount INTEGER, HispanicCount
INTEGER, PacificCount INTEGER, WhiteCount INTEGER, PRIMARY KEY(Name),
FOREIGN KEY(Name) REFERENCES School(Name) ON DELETE CASCADE);
```

```
CREATE TABLE State (StateName VARCHAR(20) NOT NULL, Population INTEGER,
CrimeRate REAL, HappinessScore REAL, Code CHAR(2), PRIMARY KEY(StateName));
```

```
CREATE TABLE Company (CompanyName VARCHAR(256) NOT NULL, Industry
VARCHAR(100), Location VARCHAR(256), EmployeeCount INTEGER, PRIMARY
KEY(CompanyName));
```

```
CREATE TABLE RateProf (ProfessorName VARCHAR(50), SchoolName VARCHAR(256),
State CHAR(2), StarRating REAL, DifficultyRating REAL, CourseID
VARCHAR(20), DepartmentName VARCHAR(256), StudentID INTEGER NOT NULL,
PRIMARY KEY(StudentID));
```

```
CREATE TABLE Major (Ranking INTEGER, Major VARCHAR(256) NOT NULL,
TotalCount INTEGER, WomenCount INTEGER, Category VARCHAR(256),
EmploymentCount INTEGER, MedianSalary REAL, PRIMARY KEY(Major));
```

Implemented Databases on GCP:

The screenshot displays the Google Cloud SQL console interface. The top navigation bar includes the Google Cloud logo, a dropdown menu for 'cs411-college-rec', a search bar, and various utility icons. The left sidebar shows the 'SQL' section with options for 'PRIMARY INSTANCE', 'Overview', 'Query insights', 'Connections', and 'Release Notes'. The main content area is titled 'Overview' and shows the instance 'college-rec' running MySQL 8.0. A chart for 'CPU utilization' is visible, showing a peak of 20% over the last 1 day. Below the chart, a terminal window is open, displaying the following MySQL commands and output:

```
mysql> show tables;
ERROR 1046 (3D000): No database selected
mysql> use colleges;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_colleges |
+-----+
| Company             |
| Diversity            |
| Major               |
| RateProf            |
| School              |
| State               |
+-----+
6 rows in set (0.01 sec)

mysql>
```

1000 rows of data in 3 tables

The image displays three screenshots of the Google Cloud SQL console, showing the 'Overview' page for a primary instance named 'cs411-college-rec'. Each screenshot includes a CPU utilization chart and a terminal window with a SQL query and its results.

Screenshot 1 (Top Left): The terminal window shows the query `SELECT COUNT(Name) FROM School;` with the result `5536`.

Screenshot 2 (Top Right): The terminal window shows the query `SELECT COUNT(StudentID) FROM RateProf;` with the result `20000`.

Screenshot 3 (Bottom): The terminal window shows the query `SELECT COUNT(CompanyName) FROM Company;` with the result `355122`.

The CPU utilization charts in all three screenshots show a peak in utilization around 12:00 PM on March 8, reaching approximately 10%.

Advanced SQL Query 1

This is a SQL query that finds the schools in states that have a **lower crime rate** than the average crime rate across the US. Order by CrimeRate in ascending order.

```
SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON  
(sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER  
BY st.CrimeRate;
```

- Join
- Subqueries

The screenshot shows the Google Cloud SQL console. The top navigation bar includes the Google Cloud logo, a search bar, and various icons. The main content area is divided into a left sidebar with 'SQL' and 'Release Notes' tabs, and a central pane. The central pane has a 'Need help connecting?' section with links to documentation and a 'To connect using gcloud' instruction. On the right, there are status indicators for 'Automated backups are enabled' and 'Point-in-time recovery is enabled'. Below this, a 'CLOUD SHELL' terminal window is open, showing a MySQL prompt. The terminal displays the execution of a SQL query that selects school names, states, and crime rates from a 'School' table, joining it with a 'State' table. The query filters for states with a crime rate less than the average crime rate across all states, and orders the results by crime rate in ascending order, limiting the output to 15 rows. The results show 15 schools, all from the state of New Hampshire (NH), with a crime rate of 2.344.

```
mysql> SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate LIMIT 15;
```

Name	State	CrimeRate
Antioch University New England	NH	2.344
Colby-Sawyer College	NH	2.344
College of St. Mary Magdalen	NH	2.344
Daniel Webster College	NH	2.344
Dartmouth College	NH	2.344
Franklin Pierce University	NH	2.344
Granite State College	NH	2.344
Great Bay Community College	NH	2.344
Keene State College	NH	2.344
Lakes Region Community College	NH	2.344
Manchester Community College (N.H.)	NH	2.344
N.H. Technical Institute-Concord's Community College	NH	2.344
Nashua Community College	NH	2.344
New England College	NH	2.344
New Hampshire Institute of Art	NH	2.344

```
15 rows in set (0.00 sec)
```

```
mysql>
```

Before indexing, the cost is **9878.54**

```
mysql> EXPLAIN ANALYZE SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;
+-----+
| EXPLAIN
+-----+
|
+-----+
| -> Sort: st.CrimeRate (actual time=4.303..4.495 rows=2245 loops=1)
|   -> Stream results (cost=9878.54 rows=9852) (actual time=0.252..3.724 rows=2245 loops=1)
|     -> Inner hash join (sc.State = st.Code) (cost=9878.54 rows=9852) (actual time=0.246..2.996 rows=2245 loops=1)
|       -> Table scan on sc (cost=5.00 rows=5912) (actual time=0.048..1.785 rows=5536 loops=1)
|       -> Hash
|         -> Filter: (st.CrimeRate < (select #2)) (cost=1.92 rows=17) (actual time=0.146..0.160 rows=23 loops=1)
|           -> Table scan on st (cost=1.92 rows=50) (actual time=0.088..0.099 rows=50 loops=1)
|           -> Select #2 (subquery in condition; run only once)
|             -> Aggregate: avg(State.CrimeRate) (cost=10.25 rows=50) (actual time=0.040..0.040 rows=1 loops=1)
|               -> Table scan on State (cost=5.25 rows=50) (actual time=0.020..0.031 rows=50 loops=1)
|
+-----+
+-----+
```

Design 1:

CREATE INDEX schoolstate_idx on School(State);

The cost decreased from 9878.54 to **808.33** which is a big improvement! And it only took 1.47 seconds to access all the rows. This index seems important to keep.

```
mysql> EXPLAIN ANALYZE SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;
+-----+
| EXPLAIN |
+-----+
| -> Nested loop inner join (cost=808.33 rows=5685) (actual time=0.160..1.470 rows=2245 loops=1)
|   -> Sort: st.CrimeRate (cost=1.92 rows=50) (actual time=0.123..0.126 rows=23 loops=1)
|     -> Filter: ((st.CrimeRate < (select #2)) and (st.Code is not null)) (cost=1.92 rows=50) (actual time=0.094..0.104 rows=23 loops=1)
|       -> Table scan on st (cost=1.92 rows=50) (actual time=0.054..0.060 rows=50 loops=1)
|         -> Select #2 (subquery in condition; run only once)
|           -> Aggregate: avg(State.CrimeRate) (cost=10.25 rows=50) (actual time=0.024..0.024 rows=1 loops=1)
|             -> Table scan on State (cost=5.25 rows=50) (actual time=0.010..0.017 rows=50 loops=1)
|           -> Filter: (sc.State = st.Code) (cost=14.96 rows=114) (actual time=0.018..0.052 rows=98 loops=23)
|             -> Index lookup on sc using schoolstate_idx (State=st.Code) (cost=14.96 rows=114) (actual time=0.018..0.036 rows=98 loops=23)
|   |
| 1 row in set (0.01 sec)
```



```
CREATE INDEX state_idx on State(Code);
```

```
yey@yey:~$ mysql> EXPLAIN ANALYZE SELECT sc.Name, sc.State, st.CrimeRate FROM School sc JOIN State st ON (sc.State=st.Code) WHERE st.CrimeRate < (SELECT AVG(CrimeRate) FROM State) ORDER BY st.CrimeRate;'
+-----+
|      |
+-----+
| EXPLAIN
+-----+
|
+-----+
|
+-----+
|
+-----+
|
+-----+
| -> Nested loop inner join   (cost=808.33 rows=5685) (actual time=0.161..1.497 rows=2245 loops=1)
|     -> Sort: st.CrimeRate   (cost=1.92 rows=50) (actual time=0.119..0.122 rows=23 loops=1)
|           -> Filter: ((st.CrimeRate < (select #2)) and (st.'Code' is not null))    (cost=1.92 rows=50) (actual time=0.081..0.096 rows=23 loops=1)
|                 -> Table scan on st       (cost=1.92 rows=50) (actual time=0.044..0.052 rows=50 loops=1)
|                       -> Select #2 (subquery in condition: run only once)
|                             -> Aggregate: avg(State.CrimeRate) (cost=0.125 rows=50) (actual time=0.022..0.022 rows=1 loops=1)
|                                   -> Table scan on State   (cost=5.25 rows=50) (actual time=0.009..0.016 rows=50 loops=1)
|     -> Filter: (sc.State = st.'Code') (cost=14.96 rows=114) (actual time=0.019..0.034 rows=98 loops=23)
|           -> Index lookup on sc using schoolstate_idx [State=st.'Code'] (cost=14.96 rows=114) (actual time=0.018..0.037 rows=98 loops=23)
|
+-----+
|
+-----+
| 1 row in set (0.01 sec).
```

Design 3:

```
CREATE INDEX crimerate_idx on State(CrimeRate);
```

After indexing, the cost is **902.11** for our query. This is worse. In fact, adding too many indexes can create overhead and slow down our performance. It seems to be more efficient to do a full table scan than create an index for CrimeRate, especially because we're only using CrimeRate to calculate the Avg(CrimeRate) and nothing else. It is not a very important/efficient index to keep.

[illegible]

In conclusion, we choose **Design 1** with the lowest cost and fastest lookup.

Advanced SQL Query 2

This is a SQL query that finds the top schools with the highest average RateMyProfessor rating.

```
SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON  
(sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
```

- Group By
- Join

```
mysql> SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC LIMIT 15;
```

Name	avg_rating
Centralia College	5
Maryland Institute College of Art	5
Post University	5
Northwestern University	5
Clemson University	5
Springfield Technical Community College	5
Dakota Wesleyan University	5
Stonehill College	5
Averett University	4.9
Ohio State University: Newark Campus	4.8999999999999995
University of the Incarnate Word	4.8999999999999995
Bethel College	4.8000000000000001
Alma College	4.8
Washington University in St. Louis	4.8
Chabot College	4.8

```
15 rows in set (0.03 sec)
```

Before indexing, the cost is **8955.20**.

```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
```

```
--+-----
```

```
| EXPLAIN
```

```
--+-----
```

```
|
```

```
--+-----
```

```
-> Sort: avg_rating DESC (actual time=33.110..33.145 rows=387 loops=1)
```

```
-> Table scan on <temporary> (actual time=0.002..0.119 rows=387 loops=1)
```

```
-> Aggregate using temporary table (actual time=32.823..32.965 rows=387 loops=1)
```

```
-> Nested loop inner join (cost=8955.20 rows=19811) (actual time=0.089..21.501 rows=13569 loops=1)
```

```
-> Filter: (rp.SchoolName is not null) (cost=2021.35 rows=19811) (actual time=0.054..8.004 rows=20000 loops=1)
```

```
-> Table scan on rp (cost=2021.35 rows=19811) (actual time=0.052..6.393 rows=20000 loops=1)
```

```
-> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20000)
```

```
--+-----
```

```
|
```

```
--+-----
```

```
1 row in set (0.03 sec)
```

CREATE INDEX rating_idx ON RateProf(StarRating);

The cost stayed the same at **8955.20**. This is not a very significant difference and the index may be unnecessary.

```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
+-----+
| EXPLAIN
+-----+
|
+-----+
| -> Sort: avg_rating DESC (actual time=32.184..32.215 rows=387 loops=1)
|   -> Table scan on <temporary> (actual time=0.002..0.096 rows=387 loops=1)
|     -> Aggregate using temporary table (actual time=31.916..32.033 rows=387 loops=1)
|       -> Nested loop inner join (cost=8955.20 rows=19811) (actual time=0.119..21.181 rows=13569 loops=1)
|         -> Filter: (rp.SchoolName is not null) (cost=2021.35 rows=19811) (actual time=0.079..7.929 rows=20000 loops=1)
|           -> Table scan on rp (cost=2021.35 rows=19811) (actual time=0.077..6.431 rows=20000 loops=1)
|             -> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName) (cost=0.25 rows=1) (actual time=0.000..0.001 rows=1 loops=20000)
|
+-----+
1 row in set (0.03 sec)
```

```
CREATE INDEX rating_idx ON RateProf(StarRating);
CREATE INDEX name_idx on School(Name);
```

The cost is now **8955.20** which shows that the cost remains the same. So the index appears to make no difference.

```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
+-----+
| EXPLAIN |
+-----+
|         |
+-----+
| -> Sort: avg_rating DESC (actual time=33.312..33.353 rows=387 loops=1)
|   -> Table Scan on <temporary> (actual time=0.003..0.106 rows=387 loops=1)
|     -> Aggregate using temporary table (actual time=33.035..33.163 rows=387 loops=1)
|       -> Nested loop inner join (cost=8955.20 rows=19811) (actual time=0.105..21.914 rows=13569 loops=1)
|         -> Filter: (rp.SchoolName is not null) (cost=2021.35 rows=19811) (actual time=0.065..8.189 rows=20000 loops=1)
|           -> Table scan on rp (cost=2021.35 rows=19811) (actual time=0.063..6.682 rows=20000 loops=1)
|             -> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20000)
|         |
|       |
|     |
|   |
+-----+
| 1 row in set (0.03 sec)
```

```
CREATE INDEX rating_idx ON RateProf(StarRating);
CREATE INDEX name_idx on School(Name);
CREATE INDEX rpschool_idx on RateProf(SchoolName);
```

The cost is now **8955.20** which shows that the cost remains the same. So the index appears to make no difference.

```
mysql> EXPLAIN ANALYZE SELECT sc.Name, AVG(rp.StarRating) as avg_rating FROM School sc JOIN RateProf rp ON (sc.Name = rp.SchoolName) GROUP BY rp.SchoolName ORDER BY avg_rating DESC;
+-----+
| EXPLAIN
+-----+
|
+-----+
| -> Sort: avg_rating DESC (actual time=33.866..33.900 rows=387 loops=1)
|   -> Table scan on <temporary> (actual time=0.003..0.116 rows=387 loops=1)
|     -> Aggregate using temporary table (actual time=33.576..33.715 rows=387 loops=1)
|       -> Nested loop inner join (cost=8955.20 rows=19811) (actual time=0.154..22.239 rows=13569 loops=1)
|         -> Filter: (rp.SchoolName is not null) (cost=2021.35 rows=19811) (actual time=0.087..8.344 rows=20000 loops=1)
|           -> Table scan on rp (cost=2021.35 rows=19811) (actual time=0.085..6.774 rows=20000 loops=1)
|             -> Single-row index lookup on sc using PRIMARY (Name=rp.SchoolName) (cost=0.25 rows=1) (actual time=0.001..0.001 rows=1 loops=20000)
|
+-----+
```

We conclude that no indexes are necessary. The default index is the simplest and efficient design.